

Retek[®] Active Retail Intelligence[™] 11.0

Operations Guide

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403
USA
888.61.RETEK (toll free US)
Switchboard:
+1 612 587 5000
Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom
Switchboard:
+44 (0)20 7563 4600
Sales Enquiries:
+44 (0)20 7563 46 46
Fax:
+44 (0)20 7563 46 10

The software described in this documentation is furnished under a license agreement, is the confidential information of Retek Inc., and may be used only in accordance with the terms of the agreement.

No part of this documentation may be reproduced or transmitted in any form or by any means without the express written permission of Retek Inc., Retek on the Mall, 950 Nicollet Mall, Minneapolis, MN 55403, and the copyright notice may not be removed without the consent of Retek Inc.

Information in this documentation is subject to change without notice.

Retek provides product documentation in a read-only-format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

Retek[®] Active Retail Intelligence[™] is a trademark of Retek Inc.

Retek and the Retek logo are registered trademarks of Retek Inc.

This unpublished work is protected by confidentiality agreement, and by trade secret, copyright, and other laws. In the event of publication, the following notice shall apply:

©2004 Retek Inc. All rights reserved.

All other product names mentioned are trademarks or registered trademarks of their respective owners and should be treated as such.

Printed in the United States of America.

Customer Support

Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
----------------	---------------------

E-mail	support@retex.com
--------	-------------------

Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
-----------------	---

Phone	+1 612 587 5800
-------	-----------------

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835

Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403
------	---

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step-by-step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction	1
Who This Guide is Written for	1
What is and is Not in This Guide.....	1
Sources of Additional Information	1
How and When to Use this guide	1
Chapter 2 – Process Overview	3
ARI Shutdown	3
Metadata Modification.....	4
Rule Construction and Modification.....	4
Import – Export Tool (IET).....	4
Code Generation	5
ARI Start	5
Summary	6
Chapter 3 – Process Details	7
ARI Logs.....	7
Understanding Log Types	7
Setting the Log Level	7
Reviewing the Log	7
Purging the Log	7
DBA_JOBS Queue	7
Scheduler.....	8
Pipe Writer	8
EVE (Exception Validation Engine).....	9
Code Generation	9
History Purge	9
Administrator Groups	9

Chapter 4 – Product Integration.....	11
Metadata.....	11
Merchandise Transaction System.....	11
Multilanguage Support.....	12
Presentation Interface.....	12
Retek Navigator	12
Retek Data Warehouse.....	12
RDW Report Setup.....	13
Metadata Setup	13
Appendix A – Architectural Reference	15
Analyst Process Definition	15
1. Set Up and Maintain Metadata	15
2. User/Group Maintenance	16
3. Schedule Maintenance.....	16
4. Exception Type Maintenance.....	16
5. Event Type Maintenance.....	16
System Process Management.....	17
1. ARI Daily Management	17
2. Table Manager and Queue Process Builder	17
3. Trigger Builder.....	17
4. Scan Code Builder.....	18
5. Validation Procedure Generator	18
6. Evaluation Procedure Generator.....	18
Exception Candidate Detection.....	19
1. Real-time Monitor	19
2. Periodic Monitor.....	19
3. Scanner Builder	20
4. Data Warehouse Monitoring	20
5. External API Monitor.....	20
6. Trickle Data Monitor.....	20
Candidate Validation and Event Creation.....	21
1. Validation Engine.....	21
2. Realm Queue Process.....	21
3. Exception Creation and Validation	22
4. Event Creation and Evaluation.....	22

User-Initiated and Automated Event Resolution	23
1. Alert Viewer.....	23
2. Event Management.....	24
3. Exception Revalidation	24
4. Event Reevaluation.....	24
5. Scheduled Reevaluation	24
6. Action Execution.....	24
Appendix B – API List	25
Error and Activity Logging.....	25
Starting the Pipe Writer.....	25
Stopping the Pipe Writer	25
Changing the Log Level	25
Stopping and Starting the Backend.....	25
Starting the Master Processes.....	25
Starting EVE Only.....	25
Stopping the Master Processes	26
Stopping EVE Only	26
Stopping All ARI Processes	26
Code Generation	26
Scheduler.....	27
Starting the Scheduler	27
Stopping the Scheduler.....	27
Signal-Driven Schedule Signaler	27
EVE (Exception Validation Engine).....	28
Starting EVE	28
Stopping EVE.....	28
Periodic Purges	28
Event Purge	28
Event History Purge	28
ARI Alert Notification API.....	29
End User Cases.....	29
Architecture	29
Implementation.....	30
Testing Scenarios	31

Appendix C – ARI Options.....	33
EVE_NUM_THREADS	33
EVE_QUEUE_REFRESH_INTERVAL.....	33
INTERNAL_SCHEMA.....	33
MASTER_SCHEMA.....	33
MAX_EVENT_RECURSION	33
REEVAL_STATUS_LOCKOUT.....	33
PRIMARY_LANGUAGE_NUMBER.....	34
ANALYST_ADMIN_GROUP_ID.....	34
CLOSE_EVENT_REALM_ID.....	34
ERROR_ADMIN_GROUP_ID	34
EVENT_INSTANCE_PARM_ID	34
EXCEPTION_CREATE_DATE_PARM_ID.....	34
EXCEPTION_CREATE_DATE_USER_ID.....	34
LOG_LEVEL.....	34
FORWARD_GENERATION_HOURS	35
RDW_LINK.....	35
RDW_OWNER.....	35
RDW_PREFIX	35

Chapter 1 – Introduction

This guide is designed to explain the administrative processes that support the operation of Active Retail Intelligence 11.0. Most of this information is not found elsewhere in the documentation.

Who This Guide is Written for

There are three key roles in the administration and use of ARI: the end-user, the business analyst and the database administrator. This guide is essential reading for both the database administrator and the business analyst who are responsible for ARI operations and maintenance.

What is and is Not in This Guide

This guide contains an overview of ARI configuration and maintenance processes with detailed focus on system administration tasks. It does not contain a detailed explanation of how to create ARI business processes.

Sources of Additional Information

Additional information of interest to database administrators can be found in the ARI Installation Guide. Additional information for business analysts can be found in the ARI Installation Guide as well as in the User Guide/Online Help. ARI Process modification methodology and detailed technical training is available from Retek.

How and When to Use this guide

None of the processes described here should be undertaken until after ARI installation is complete. Appropriate administration of ARI is critical to its successful operation, and although many of the tasks, once in a production, are primarily the responsibility of the database administrator, significant collaboration between the database administrator and business analyst is required throughout the rule definition process. Database administrators should read and understand this guide after installation and before doing any additional ARI work. Business analysts should read and understand this guide before creating any ARI rules. One of the first things to do after reading and understanding this guide is to make sure the ARI Options table values are all set correctly (see Appendix C for details of the options).

Chapter 2 – Process Overview

As discussed in the Overview sections of the online help, ARI is only a tool that helps with several steps of a larger process, which, for context, is reviewed here. This larger process, to implement a new business process, or existing process modification, involves first gathering requirements and conducting user walkthroughs, designing possible implementation strategies, reviewing impact, choosing a strategy and conducting additional walkthroughs. This is followed by development of rules, test implementations and user acceptance. Finally, the rules are moved into production. The sub-process relevant to ARI administration is essentially the same in development, test and production.

More than a single, specific process, the sub-process relevant to database administrators is a set of processes that can be linked in a number of different ways depending on the operational environment. Factors such as whether the database is shut down regularly or not, and how the shutdown occurs (shutdown immediate or otherwise), will impact exactly how these sub-processes should be implemented.

ARI Shutdown

ARI uses the DBA_JOBS queue to control the background processes essential to exception detection and validation. ARI also uses generated code in these detection processes. (A more detailed architectural overview can be found in Appendix A.) Before making *any* DDL changes in any ARI schema or in any schema monitored by ARI, including recompiling packages, alter tables, creating new functions, or even running the ARI code generator, it is critical that the Exception Validation Engine (EVE) be halted from processing exception candidates. (Stopping EVE can be done using one of the many stop methods described along with the other APIs in Appendix B.)

EVE uses persistent variables to control multi-threading and other aspects of its operation. Because of this, EVE should be stopped before bringing down the database and restarted when the database is brought back up. If EVE is not stopped, either due to an unexpected shutdown or shutdown before EVE is cleared from the DBA_JOBS queue (EVE may take a few minutes to finish processing after the shutdown script is run), the shut down request should be sent immediately after the database is restarted. Once it has cleared the DBA_JOBS queue, EVE can then be restarted.

Metadata Modification

As a rule, metadata should be synchronized with the actual DDL and other objects (Oracle Forms, for example) it describes at all times. Since such instantaneous synchronization is essentially impossible, the next best thing is that these tasks are performed in immediate sequence. This is not terribly difficult since the DDL and Forms should change very little during production, and when they do, non-administrative users usually must be logged out while the DDL changes are taking place. Note that while the DDL and metadata changes are being made, EVE should NOT be running.

In series with actual DDL changes and metadata modification, the code generator must also be run immediately while database users are still logged out. DDL changes can invalidate generated code, but the generated code can only be corrected accordingly after the metadata is changed, so this third thing, code generation, must be done in sequence with its precedents. EVE should not be running during code generation either, as will be discussed.

A possible exception to this rigorously limited circumstance under which metadata can be changed is when new actions or functions are added to the system. By the time you have gone through development and test, you should have the expectation that migration of the new packages or Forms will be routine. Creating the metadata for these objects before shutting down EVE and doing the other two steps in the process (a) actually installing the changes and (b) running the code generator. If you choose to make modifications to metadata at times other than when EVE is shutdown it is critical that you only add or change just added metadata and that you actually add the new code before next running the metadata generator.

Rule Construction and Modification

Rules can be constructed or modified at any time except while the code generator is running. New rules appear in the system only after the code generator is run. Modifications involving an exception or event end date require that the code generator be run before they will take effect. Other modifications, such as the linking between exceptions and event or schedule to either exceptions or events are dynamic and will take effect as soon as they are applied. There is no significant administrative task here, but this process is highly dependent on the code generation process that it is worth highlighting. Rule construction drives the need for code generation, so database administrators and business analysts may like to stay coordinate about how and when new rules are likely to be created.

Import – Export Tool (IET)

IET is a tool for copying rules into and out of an ARI instance. It allows ARI consultants to supply clients with pre-packaged rules. It allows clients to replicate rules between different testing and production instances. This is preferable to manually recreating rules within each ARI instance. Rules are exported to .xml documents in a database-independent form. From these documents the rules can be imported to a different ARI instance, where IET will attempt to resolve metadata references within the rule to corresponding metadata in the new ARI instance. ARI consultants can also package up supporting actions and data that will be added to new ARI instance with the rule during import.

Code Generation

Two processes require code generation. Rule construction/modification, and metadata changes. After either of these occurs, code generation is required. Code generation is required immediately after metadata changes (except in the instance of new metadata where delayed code generation may be deferred), and in a reasonable timeframe after rule construction/modification. (For details on executing code generation, see the API details in Appendix B.)

One of the issues with when to run code generation is how soon it needs to be run after rule construction or modification. The code generator builds supporting code for exceptions and events for all new exceptions and events configured to start before some future time; specifically, now (the time the code generator is started) plus the number of hours defined by the `FORWARD_GENERATION_HOURS` option. (For forward generation details reference Appendix C on configuring ARI options).

Obviously code cannot be generated constantly, so forward generation is done in anticipation of a business analyst defining an exception or event several hours, or even a few days, before they actually want it to take effect. Generally, significant planning and design effort are put into defining exceptions and events, so planning well enough ahead that the code generator need not be run immediately is typical in a production environment. Certainly the code generator can be run at any time that EVE is stopped, but ideally in a production environment it is run at an off-hours time so that any issues can be handled with minimal impact on the production environment.

Exceptions and events can be linked even after code generation, both linked to each other and to schedules. However, rules cannot be modified after code generation, and the consequence of forward generation is that changes to effective dates do not take effect until the code generator is run again. An exception or event set up to start immediately will only start as soon as the code generator is rerun. One scheduled to stop in three weeks that you want to stop now instead will only stop as soon as you set the end date and rerun the code generator.

Both of these cases can be handled as they occur by simply stopping EVE and running the generator on demand, but a typical configuration might be to forward generate by 30 hours and run the generator daily at approximately the same time (as a scheduled job even) every day. This typical configuration does not preclude on-demand generation as well (alternatives and other issues are in Appendix C).

ARI Start

Once the first rules are built and the code generator run, EVE and the other continuously running processes (Scheduler and Pipe Writer) should be active whenever the database is running. Running the appropriate ARI start program can start all of these programs. If only EVE is shut down (perhaps during a code generation run) and in need of starting, then a different program can be used to start EVE only (ref. Appendix B).

Summary

The following diagram summarizes 4 possible options (vertically) for performing key routine ARI tasks. The critical processes for which EVE is not running and users are logged out are highlighted in darker boxes, specifically Change Metadata and Modify DDL. The left-hand column shows the status of the DBA_JOBS queue each step of the way.

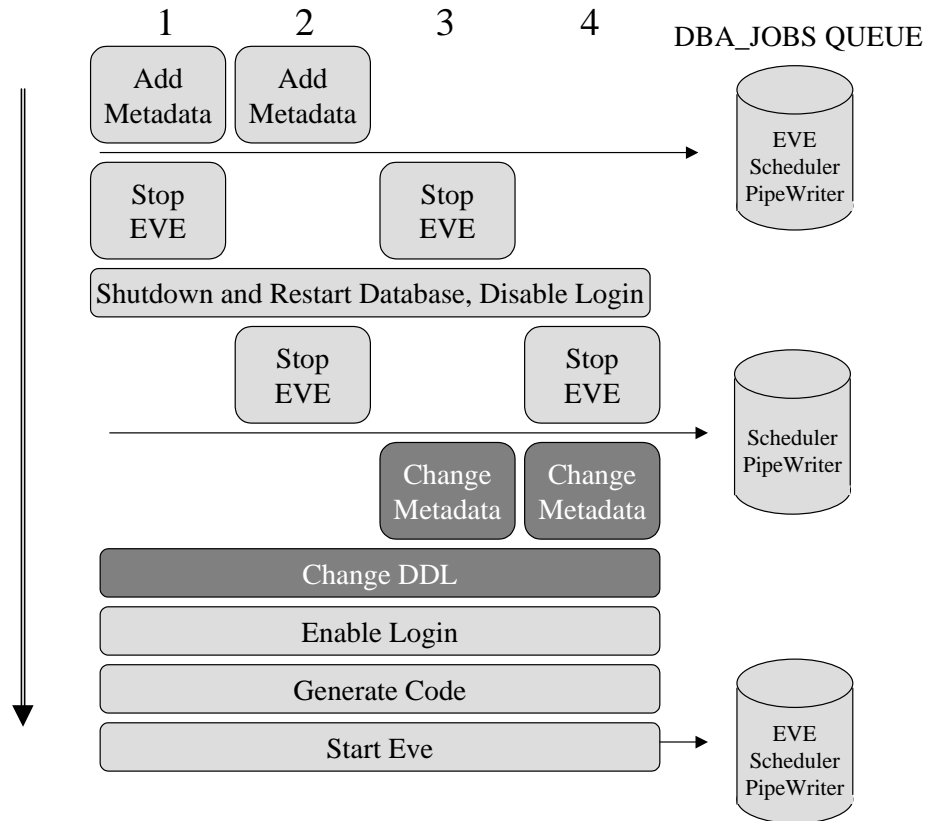


Diagram: DBA_JOBS Queue

Chapter 3 – Process Details

ARI has several continuously running processes whose progress should be checked periodically. The processes write to an activity log and an error log table, `ARI_ACTIVITY_LOG` and `ARI_ERROR_LOG` respectively. The continuously running processes and the additional processes they spawn are queued in the `DBA_JOBS` queue. Both business analysts and database administrators will need to be familiar with these tables and processes and at least the database administrator, with some escalation procedure to the business analyst, will want to review the tables and processes on at least a daily basis.

ARI Logs

Understanding Log Types

`ARI_ERROR_LOG` should usually be empty. When it isn't, then the errors should be reviewed. Certain types of errors may be non-critical, but a repeated number of such errors may indicate some action is required. Everything written to the error log is also in `ARI_ACTIVITY_LOG` (the activity log), but the activity log contains many more details of what the system is doing and can be helpful for troubleshooting.

Setting the Log Level

ARI backend processes have message levels assigned to log entries. Level 1 messages are the most important with level 3 messages being at a fine detail level. Typically we only run level 2 at Retek, but level 1 may be sufficient for production. The log level is cached for a given database session, so to change it requires starting a new session. In the case of EVE or the Scheduler this means stopping them, resetting the `LOG_LEVEL` option on `ARI_OPTIONS` (see Appendix C) and creating a new database session when restarting them.

Reviewing the Log

When the error log is empty the activity log usually does not need review (unless things are not happening that should be, in which case lack of recent entries in either log may indicate ARI is not running). When any errors have been debugged the logs can be truncated.

Purging the Log

There is no automated process for purging the error and activity logs, however, these tables can become quite large if they are not truncated periodically. This truncation is at the discretion of the business analyst and database administrator reviewing the logs. In production daily truncation after review should be a standard procedure.

DBA_JOBS Queue

`DBA_JOBS` will usually contain a scheduler job, a pipe writer job, an EVE job and a number of validation threads (spawned by EVE) and a number of batch scans and event reevaluation jobs (spawned by the scheduler). Broken jobs indicate something not working correctly. Broken validation threads typically mean that EVE is not going to be spawning as many threads as it could because it will be waiting for the broken thread to return. After troubleshooting, broken jobs should be removed (`DBMS_JOB.REMOVE`) and, if any of the jobs were validation threads, EVE should be shutdown and restarted.

Scheduler

The scheduler lives in the DBA_JOBS queue. Every 5 minutes it determines what ARI schedules are due for execution and generates batch exception scans and event reevaluation blocks and inserts them into the DBA_JOBS queue. If an exception has multiple schedules and is already being scanned by a still queued function, that exception will not be added to the new scan if it is scheduled again. The scheduler should remain in the DBA_JOBS whenever the database is running after ARI is installed. It is not harmful for the schedule to be out of the queue, but scheduled jobs will get backed up until it is restarted and will not run on schedule.

Multiple instances of this job should not be running. You must decide from a scheduling standpoint whether to remove it at shutdown (and again at startup if the typical shutdown is a shutdown immediate) or whether just to leave it in the queue past shutdown (it should start up when the database restarts). In the latter case the DBA_JOBS should be checked daily to make sure it is still an active job (this can be validated by an entry in ARI_ACTIVITY_LOG approximately every five minutes indicating the scheduler is running looking for things to do).

Pipe Writer

The pipe writer reads information off the ARI error pipe and writes it into the appropriate activity or error table. This pipe is used to send error information from programs where we do not want to execute a commit to get error or activity information. The pipe writer should be running whenever the database is running. The consequences of its not running will be missed log information and perhaps, eventually, a full pipe which may cause other errors. In terms of managing this job, it has the same issues as the schedule as to when and how it can be monitored – though it doesn't log anything accept incoming messages themselves. DBA_JOBS stay queued through a database shutdown and should resume when the database is restarted.

EVE (Exception Validation Engine)

EVE uses the REALM_QUEUE_CTL table to track which realms it is currently processing and which ones to do next. EVE creates validation threads that work on individual realms. These threads get added to DBA_JOBS and when they are complete they tell EVE they are finished so that even can update the control table and create a new thread on a different realm. Throughout all of this, EVE and its validation threads write to the log extensively.

EVE and its validation threads should be removed from the DBA_JOBS queue before shutdown, or, if they are not because the shutdown is immediate, they should be removed on restart. This is necessary because EVE uses memory persistent information to control its threading process. The issue is primarily one of conserving system resources, though other undesirable behaviors can also result from having the too many additional validation threads that EVE creates when not properly stopped completely either before shutdown or immediately after restart. Before shutting down or running the code generator, it is important to check the DBA_JOBS queue to make sure that EVE is actually done, since its shutdown procedure may not be immediate (the shutdown signal is sent on a pipe which is why the stop procedure may return immediately).

Code Generation

The code generator has already been covered extensively. Details of when and how it can be used are that EVE should not be running and the metadata and DDL and Forms must all be synchronized when it is run. Provided these conditions are met it can be run anytime, though it is generally recommended that it be scheduled periodically and during off-hours to reduce the impact of any code generation errors that might occur. The code generator logs heavily. For troubleshooting it is sometimes helpful to stop both the pipe writer and the scheduler during code generation just to simplify reading the log activity.

History Purge

The history purge process should be scheduled often, perhaps even daily. Unless it produces an error it should not require any special attention. History purge removes events older than the history retention days specified in the event definition.

Administrator Groups

ARI uses to special user groups, an error group and an analyst group. These groups must always have at least one user each. Both the key database administrator and business analyst should be added to each group before any rules are put into production. If not these users, then someone who will be involved in the daily administration of ARI *must be added before building any rules*. The error group is the assignment target group for all events that enter an error state, the analyst group is the assignment target group for all events that are assigned to otherwise empty groups!

Both groups may also be useful for additional analyst and error monitors that you might like to create with ARI (or that Retek might in some future release deliver or help build from standard templates during a consulting engagement). The analyst group might be used to build a security function for a custom modification to enforce application security on access to the ARI administrative forms. (Note that such access level modifications are allowed in spite of the general caveat that all custom modifications of ARI are unsupported – such a modification is not strictly a functional modification.)

Chapter 4 – Product Integration

ARI is a tool intended for use with one or more applications, so integration with other products is an inevitable post-installation configuration requirement for useful production operation. Generic integration issues include creating metadata for the systems ARI will work with, linking ARI into the main application presentation interface of these systems (or not), and multi-language support. Some specific assistance is provided to simplify integration with Retek's Merchandise Transaction System (MTS) and the Retek Data Warehouse (RDW).

Metadata

Once the seed data has been installed, metadata needs to be created for the systems that ARI will work with. Metadata is the foundation of all ARI rule building, so it is critical that it be accurate. Metadata maintenance is a database administrator task that should become part of the routine that goes with changing Form specifications or table definitions. Fortunately, this doesn't occur very often in a production system, so it should not be a particularly time consuming task.

Metadata maintenance must be done in sync with DDL changes. Users should not be in the system trying to work with ARI Alerts during the gap in time between when DDL changes are made and when the system is updated. Furthermore, EVE cannot be running during this time. The exception to this rule is when metadata is being added to the system either initially or even after production use of the system has begun. In this case, the metadata can be added during production hours provided the code generator is not run until after the DDL changes are made. This means for initial setup of metadata the entire process can be done during production (and you don't have to log users out for 3 days while you do it).

Note that it is not necessary that the entire system be described in metadata, just that the metadata that is described needs to be accurate. This can be a time saver since you only need to create metadata for the rules you plan to build. There may be some tables in the system that you will not ever need to select data from or monitor, and there will be even more Forms and PL/SQL actions that you will not need. Analysis before implementing an ARI rule should clearly identified what physical data entities and actions will be needed, so creating appropriate metadata, if it wasn't already created for another ARI rule, could be treated as part of the rule development process.

Merchandise Transaction System

For owners of the Retek Merchandise Transaction System version 9.0 Retek has pre-defined the metadata for all of the tables in the system. This metadata can be populated (provided it is the first metadata added after running the seed data scripts that are required in the ARI Installation Guide) by executing in order the scripts provided: mts_parm_type.sql, mts_realm.sql and mts_parm.sql.

Multilanguage Support

ARI has a few text fields that are translatable that appear in the end-user interfaces, such as the event type name and state name, and the names of the parameters used on the event. ARI can also translate text string data values that may be included as data values in an event, provided these data values have translation strings in the ARI dictionary. However, having these strings in the ARI dictionary is unlikely. After trying to use its internal dictionary, ARI also calls an external notification API that can be configured to call translation methods from other applications, in particular the methods used for the external (to ARI) applications that are the data values' systems of origin. (See Appendix B for more API details).

Presentation Interface

A key feature of ARI is to be able to be notified in your operational applications that an alert has occurred in ARI. To that end ARI provides an API set that will enable you to put a button on the toolbar of yours Forms applications. The button can represent whether any new (undelayed) alerts exist for the user logged into the database, and pressing it will launch the ARI Alert Viewer. For non-Forms applications it is possible to bypass the Forms library and simply access the PL/SQL procedure that does all of the work. (For details on the API, consult Appendix B.)

ARI has a startup form (aristart.fmb) that allows access to the ARI forms. This form is primarily intended for administrative, full functionality access users to get into ARI. The integrated end-user entry point into the system will typically be through a notification button on the toolbar of the applications that ARI is monitoring, as just described. However, if that entry point is not used, you likely will want to add role-based security to the menus in the ARISTART form to restrict end-user access.

Note that the ARI launch/notification button already exists in the Merchandise Transaction System, and the API interface is already installed. To make the interface operational, simply drop the PL/SQL portion of the interface (ARI_INTERFACE_SQL) from the MTS product schema and replace it with a synonym to that same package in the ARI schema.

Retek Navigator

The Retek Merchandise Transaction system start up form (after the login splash screen) is the Retek Navigator. An alternative to using the ARISTART form for any administration is to add the ARI forms to the Retek Navigator (and then adding appropriate role level security etc). The syntax for the calls to the ARI forms is easily extracted from the ARISTART menu. Consult the Merchandise Transaction System user guide for Navigator configuration details.

Retek Data Warehouse

ARI has monitoring and metadata methods specific to the Retek Data Warehouse. It is not a typical Oracle transaction system and so the integration with this product has been done above and beyond typical ARI functionality. Except for creating a database link from the ARI Master schema to the Retek Data Warehouse instance, no special configuration of ARI is necessary to use this feature. If you do not have the RDW, nothing special need be done.

RDW Report Setup

The RDW Interface accesses RDW report data that is stored in a Datamart. The Datamart functionality allows report data to be stored in Oracle tables that can be accessed by ARI. Reports that are to be run for ARI must be set up to store their data in a Datamart schema. For more information about setting up reports to store to a Datamart, refer to the following documentation at the MicroStrategy support web site <https://support.microstrategy.com/home.asp>, MSTR Tech Note #TN5200-071-0082. This schema must either be located on the ARI database, or on another database (such as the RDW database) which is accessible through a database link from the ARI Master schema. Moreover, although in a typical RDW installation some reports may be run and cached periodically while many others will be generated on demand, reports monitored by ARI are generally scheduled. This RDW report scheduling usually involves some manual synchronization with the exception's report-monitoring schedule, configured in ARI. All of these factors lead us to recommend that reports intended for monitoring by ARI be set up under a single user in a single project, even if some of these reports are redundant with others that already exist in the RDW.

Metadata Setup

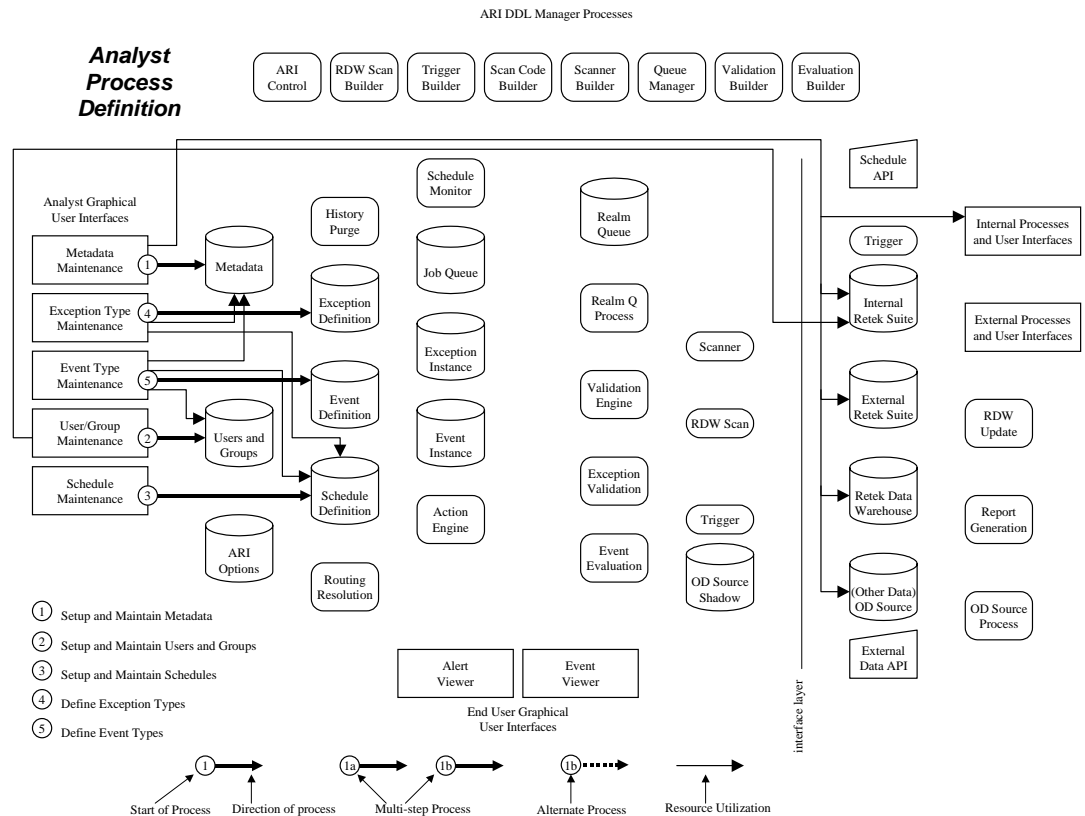
Metadata auto-discovery will only search for reports owned by a single Datamart schema, in a single database, which are stored in the ARI options RDW_OWNER, RDW_LINK. Auto-discovery stores these option settings in the SCHEMA and DB_LINK fields on the table REALM, and it is these fields which are used when searching for cached reports during exception monitoring. Therefore it is possible to manually create metadata for reports stored in a different schema and/or database. In addition, if the ARI option RDW_PREFIX is set, auto-discovery will only search for Datamart tables beginning with this prefix. During auto-discovery, the RDW Interface must search for Datamart tables created for a report, in other words, the report must already have been run and stored a non-empty result set to the Datamart before auto-discovery can be executed.

Appendix A – Architectural Reference

This section provides diagrams and descriptions of the processes that drive ARI. These processes are grouped according to the main functional uses of ARI.

Analyst Process Definition

Before ARI will perform any functions, several processes must be completed to tell ARI what to do. The following diagram and description walk through these processes.



Process Diagram: Analyst Process Definition

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. Set Up and Maintain Metadata

Database administrators keep metadata and Oracle DDL synchronized. In a production environment, the DDL should seldom if ever change. So, once set up, maintaining the metadata is not a time-consuming task. Metadata for new functions/actions to support ARI processes defined by business analysts can be set up and maintained by either a DBA or a business analyst. Metadata for Data Warehouse reports and external systems (those using the API) is setup and maintained by business analysts. Like functions and actions, these reports and external systems are probably defined in metadata on-demand, as needed to support specific ARI processes.

2. User/Group Maintenance

Performed by business analysts, user and group maintenance supports event assignment and event supervision assignment. Users entered here must be defined as Oracle users in the database.

3. Schedule Maintenance

Schedules indicate when a task will occur. These are maintained by business analysts and assigned to exception and event definitions to determine when periodic exception scans and automatic event re-evaluations will occur.

4. Exception Type Maintenance

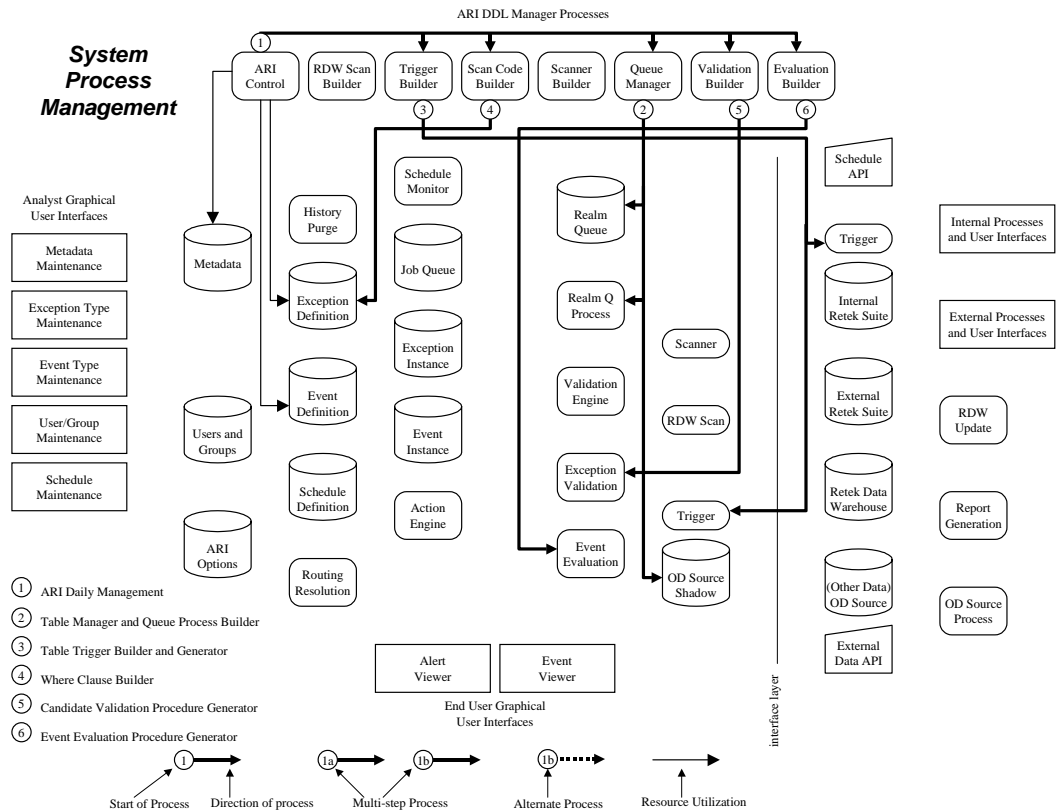
Business analysts maintain exception definitions. They define when to monitor for specific data conditions, what the minimal actionable data conditions are, and what event(s) should be created to help resolve the exception.

5. Event Type Maintenance

Also performed by business analysts, event definition maintenance determines the presentation of exception-related information in the form of alerts. An event contains an alert definition and an exception link, plus information about which users are notified, what actions are available to resolve the event, and what new data conditions will determine that the event is, in fact, resolved.

System Process Management

Using analysts' definitions, ARI prepares to monitor exceptions and present events.



Process Diagram: System Process Management

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. ARI Daily Management

The daily ARI control program is run at the beginning of the batch window. After all users are cleared, it shuts down the currently active ARI processes, rebuilds exception and event management code (processes 2-6), and then restarts the ARI processes.

2. Table Manager and Queue Process Builder

This process creates a realm queue table for every monitored realm (if one does not already exist) and deletes those realm queue tables no longer needed. It also builds/drops (as appropriate) realm queue procedures, one per queue, to manage queued data. Finally, it builds the shadow tables that hold data input from external (non-Oracle) data systems.

3. Trigger Builder

Drops and creates triggers to monitor real-time exceptions on the Oracle retail suite that is part of the same DDL as ARI. Also drops/creates triggers on the external source shadow tables that make externally sources data not real-time monitored but trickle-monitored. (The data is moved into the queue real-time as it is received).

4. Scan Code Builder

The scan code builder prepares appropriate clauses to select only rows of interest (based on the exception definition) from tables that are monitored via periodic scanning. This data is stored with the exception definition for easy retrieval by the Scanner builder.

5. Validation Procedure Generator

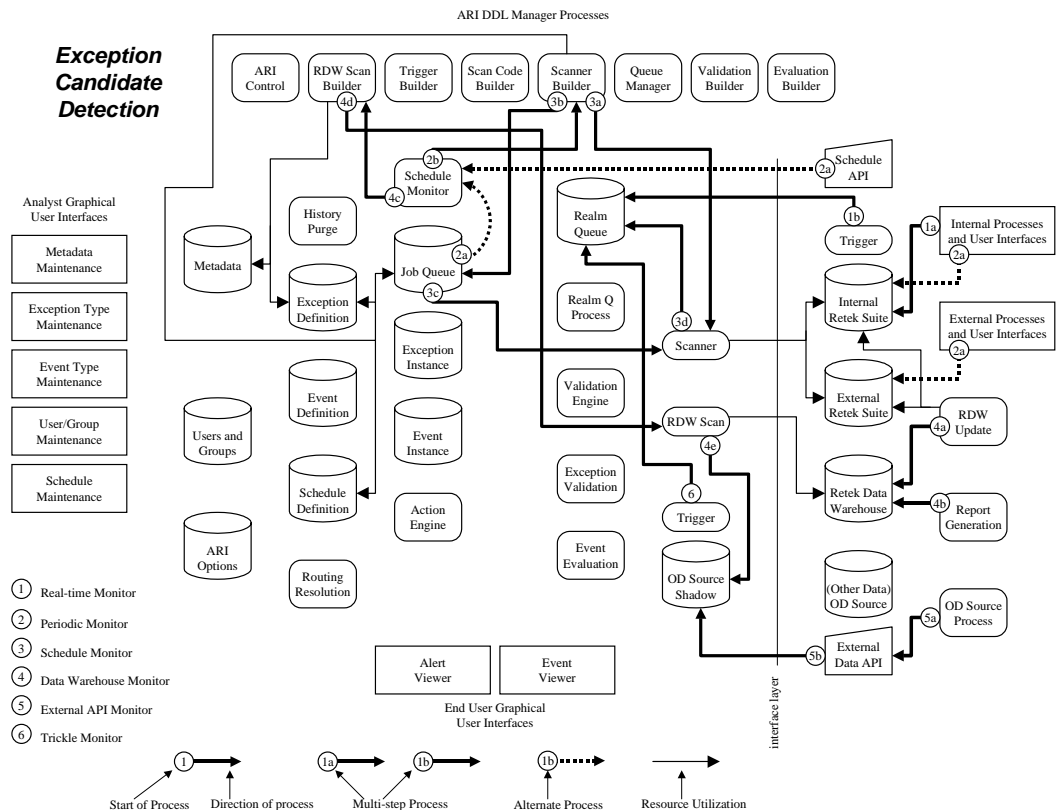
Exception candidates must be validated to determine whether an exception exists. This involves selecting additional information from other parameters and processing the conditions on those parameters. The procedure generator creates this business logic in a static package.

6. Evaluation Procedure Generator

Once an exception candidate is known to exist, it can create one or more events. Underlying data conditions may change the way an event should be presented (what actions are available) or who should be dealing with it. The reevaluation logic is encapsulated in a static package.

Exception Candidate Detection

To minimize performance impact on other systems, ARI first identifies candidate exceptions by imposing some of the exception conditions on the data set being monitored. This is done in order to filter out those that could be exceptions, based on complete condition evaluation after additional information is fetched, from those that could not be data states of interest, irrespective of what other information might be gathered up during the exception process. Candidates are queued and processed as systems resources are available.



3. Scanner Builder

The scanner builder assembles the scan code for all exceptions scheduled to occur simultaneously and places a job in the Job Queue to run those scans. As threads become available for processing, the Job Queue initiates the scans that populate data representing exception candidates into the realm queue tables.

4. Data Warehouse Monitoring

On a periodic basis data in the Data Warehouse is updated and reports are re-run. In sync with that schedule should be data warehouse monitoring, so in addition to kicking off the scanner builder the schedule monitor kicks off the RDW scan builder when appropriate as well. The scan builder dumps report data directly into a shadow table.

5. External API Monitor

This API is used to feed in data from an external (non-Retek) system. As the API is called from the external system it writes data to an appropriate shadow table within ARI.

6. Trickle Data Monitor

The trickle data monitor takes data as it arrives from non-Oracle systems via an appropriate monitor, and pre-filters it, putting the candidates into the appropriate realm queue. Although Data Warehouse monitoring is periodic, the idea of trickle monitoring is that the data is processed as soon as it arrives, which in the case of external systems depends on where in the external system the ARI External API is implemented, not on any scheduling done with ARI. Indeed, the same API could be used with the Data Warehouse, but for convenience the link to this Retek system is provided in the form of a periodic link, which should be sufficient if synchronized with the periodic updates of the Data Warehouse and its monitored reports.

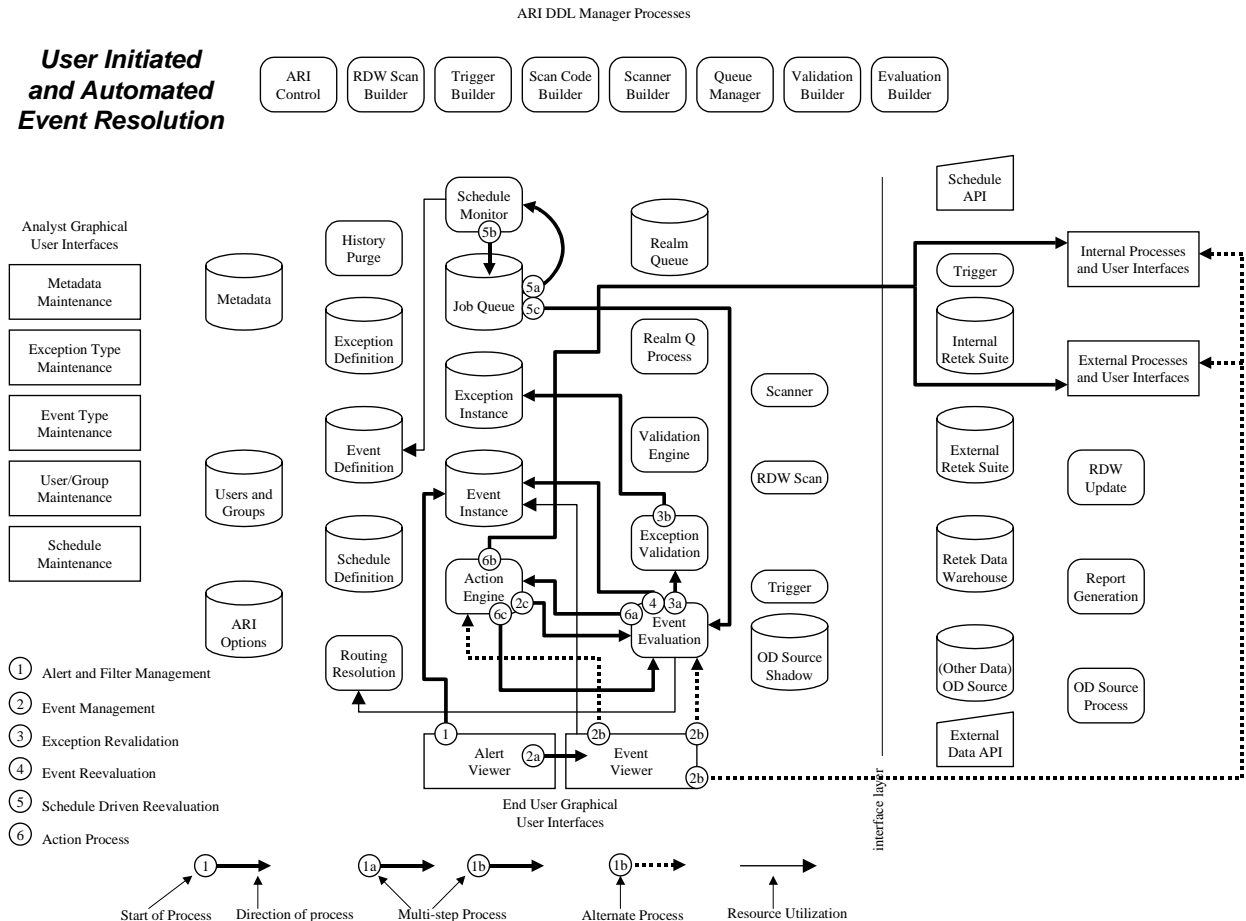
3. Exception Creation and Validation

Valid candidates become exception instances, and then event evaluation is invoked to determine which events should be created or, if they already exist, re-evaluated.

4. Event Creation and Evaluation

Event instances for each event type linked to the exception instances type must be either refreshed, if an event with the same key values already exists, or created. Evaluation occurs to determine alert routing. Then, the instance is created and, if the appropriate rules are met for a particular instance, an action may be taken automatically.

User-Initiated and Automated Event Resolution



Process Diagram: User Initiated and Automated Event Resolution

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. Alert Viewer

End users initially view events as alerts, which are just event summaries, in the Alert Viewer. This viewer allows sorting and categorization by event type, state, priority, occurrence data, and even whether the user has deferred (marked as having been viewed) the alert. In the Alert Viewer, events are grouped by type state and priority with a count of how many of each type of state and priority exist.

2. Event Management

The Event Viewer shows all events of a single type and state at any given time, and also shows all the details of those events. The user initially is shown the most recently refreshed event data (which may or may not be very recent, depending on whether it is done periodically as defined in the event type). The user may choose to reevaluate the events before proceeding, since re-evaluation may show that some events are already resolved. The user may also drill into the monitored systems to view even more information than what is presented on the event. In addition, the user may take an action to resolve the event or move it along in its process. If an action is taken, the event is still reevaluated beforehand to ensure that the event is still in the state in which the action is valid.

3. Exception Revalidation

Because the data source for many event parameters is an exception, before an event is reevaluated, its associated exception instances are revalidated. This process is just like validation, except that the exception itself is treated like a new candidate and is removed from the event and deleted if it is found to be invalid.

4. Event Reevaluation

Once the exceptions are revalidated, event reevaluation refreshes the event parameters and processes its defining rules to determine whether the event is resolved and should be closed, whether it is in the same state as it was in, and to whom it should be assigned and with what priority. If the reevaluation is user-requested either directly or as a consequence of the user attempting to take an action, and if the state and user assignment are unchanged, then the action can be executed, or if user requested, the user may continue to view and act on the event. Because the Event Viewer shows only a single state at a time, events that change state, although the same user may still own them, will not continue to be seen in the view of the event's previous state.

5. Scheduled Reevaluation

Evaluation can be user initiated via one of the previously described processes and may also occur on a scheduled basis. As with scheduled scans for exception candidates, the Job Queue notifies the scheduler to look for tasks that need to be done, which in turn places the tasks in to the Job Queue. As threads become available, these evaluation jobs, specific to an event type, are started. Each job loops through all open instances of their event type and initiates the reevaluation process.

6. Action Execution

Actions are always executed after an evaluation occurs. If an action is user-requested, the event is always evaluated first. Alternatively, as a result of an evaluation, whether prompted by a user request or a schedule, an action may be taken automatically if the event is so defined. After the action is taken, the event is always reevaluated, which may prompt another action, and so on. A well-defined event should not ever get into an infinite loop, but it is theoretically possible. Therefore, checks are in place to prevent an infinite evaluation-action-evaluation loop from occurring.

Appendix B – API List

Error and Activity Logging

There are three PL/SQL APIs related to error and activity logging that can be incorporated into scripts or used in other contexts.

Starting the Pipe Writer

`ari_error_logging.start_pipe_writer` starts the pipe writer by placing the `ari_error_logging.write_pipe` procedure into the `DBA_JOBS` queue. It must be executed while connected to the database as the ARI master schema user to ensure that the process will have all the appropriate privileges it needs while executing. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Stopping the Pipe Writer

`ari_error_logging.stop_pipe_writer` stops the pipe writer by removing the `ari_error_logging.write_pipe` procedure from the `DBA_JOBS` queue. It must be executed while connected to the database as the ARI master schema user to ensure that the job is removed from the queue. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Changing the Log Level

Within a particular database session, calling `ari_error_logging.set_log_cutoff_level` can change the log level. The set value will override the ARI options setting within that session unless the function is called again. This can be a useful debugging tool.

Stopping and Starting the Backend

The ARI Backend includes three master processes: the Scheduler, EVE, and the Pipe Writer. These processes live in the `DBA_JOBS` queue from which they run periodically. The Scheduler also creates Batch Scans that it adds to `DBA_JOBS` and EVE creates Validator Threads. EVE manages the Validator Threads and the Batch Scans manage themselves (by simply disappearing from the queue when they are done executing). There are several start and stop methods for the master processes that may be added to database start and stop scripts or executed on a regular schedule, as detailed in the Process Overview.

Starting the Master Processes

`ari_control_sql.start_ari` calls the start methods for EVE, the Pipe Writer and the Scheduler. It must be run as the ARI master schema owner to ensure that all of the processes will have appropriate privileges when they execute. This is typically placed in a start up script, or, if you prefer to let the Scheduler and the Pipe Writer simply live in the queue at all times, it may only be executed very rarely with the second start method (Start EVE) being the one that is scheduled.

Starting EVE Only

`ari_control_sql.start_eve` calls the start method for EVE. It must be run as the ARI master schema owner to ensure that EVE will have appropriate privileges when it runs.

Stopping the Master Processes

`ari_control_sql.stop_ari` calls the stop methods for EVE, the Pipe Writer and the Scheduler. It must be run as the ARI master schema user to ensure that the jobs are actually removed from `DBA_JOBS`. This is typically placed in the database shutdown script, though often daily bouncing just uses a shutdown immediate, so it is placed in the database start script instead to ensure appropriate shutdown before EVE and the other master processes are restarted.

Note that Validator Threads must have completed and exited the queue (which can have a slight delay after the Stop Process is run) before EVE is restarted. When EVE is terminated it signals the Validator Threads to abort and exit the queue, but it can (rarely) take the threads a few minutes to acknowledge the signal. This delay may be either that (a) they are queued but not yet running, in which case they must first start running as other threads are killed and become available, or (b) they are in the process of opening a cursor.

Also note that Batch Scans are not removed by this script, which is generally what is desired since Batch Scans removed from the `DBA_JOBS` queue mean that those exceptions contained in the scan will not be scanned again until their next scheduled date. Batch Scans can and should survive database outages. Individual scans can be removed using `DBMS_JOB.REMOVE` if their execution is meant to be halted (the corresponding database server process will have to be killed as well).

Finally note that if all you typically do is start and stop EVE, leaving the other master processes running, then you will want to schedule the second stop method (that stops EVE only).

Stopping EVE Only

`ari_control_sql.stop_eve` calls the stop method for EVE. It must be run as the ARI master schema owner to ensure that it will remove the process from `DBA_JOBS` correctly.

Stopping All ARI Processes

`ari_control_sql.stop_all_ari` is the same as stopping the master processes, but also kills all of the Batch Scans in the queue. In production this is almost never used, because you does not want to lose those scans, this is more of a development utility. During development an undesirable scan or set of scans may be created by mistake, in which case killing all processes may be the simplest thing to do, rather than removing them one at a time. Often this is done in conjunction with a quick database shutdown and restart to save the trouble of finding the appropriate database server processes to kill, which may be necessary since jobs in the queue once started run even if they are removed from the queue.

Code Generation

Along with EVE, this is the most critical process to ARI. The code generator must be run as the ARI master schema user to ensure that all of the code can be created properly. Also, it must only be run when EVE is and its Validator Threads have all been removed from the `DBA_JOBS` queue. Other operational issues associated with code generation are discussed throughout the Process Overview, such as scheduling it on a periodic basis or running it on demand. Re-summarizing, code generation is tied up very closely with DDL changes and EVE. Whereas users cannot be logged in during DDL changes, they can usually be logged in during code generation. The exception to this is if metadata updates to reflect the DDL changes impact any database objects used by any exceptions or events, in which case re-enabling user login after DDL changes must wait until after metadata changes are made and the code generator re-run.

Scheduler

The scheduler checks to see which scheduler are currently due to execute and creates Batch Scans and anonymous Event Revalidation Blocks to be added to the DBA_JOBS queue. The scheduler is typically started and stopped via one of the many ARI start/stop control processes, but those methods can be accessed stand-alone for custom scripting etc. The scheduler also contains an API for sending signals to set the next execute date for signal-driven schedule to the current date and time, thus causing the schedule to be executed the next time the scheduler checks schedules due for execution (approximately once every five minutes).

Starting the Scheduler

`ari_auto_scheduler.start_scheduler` add the `check_schedule` procedure (the Scheduler) to the DBA_JOBS queue. This procedure must be run as the ARI master user to ensure that `check_schedule` has the appropriate privileges when it executes. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Stopping the Scheduler

`ari_auto_scheduler.stop_scheduler` removes the `check_schedule` procedure from the DBA_JOBS queue. This procedure must be run as the ARI master user to ensure that `check_schedule` is correctly removed. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Signal-Driven Schedule Signaler

This API can may be added to other programs so that exception detection and/or event reevaluation can be attached to some other thing happening in the operational system. Each signal driven schedule has a signal text string. This string is passed in the appropriate argument of the `ari_schedule_sql.accept_signal` function to update the next execution date of the schedule associated with the signal text to the current date and time. This then causes processes associated with the schedule to be executed when the scheduler checker next executes. Note that this function contains a commit, so the likely placement of it in another procedure is just after a successful completion and commit of that procedure.

EVE (Exception Validation Engine)

EVE is the key threading process that governs ARI exception processing. It is typically started and stopped through one of the many ARI start/stop control methods, but the direct APIs for this functionality exist as well. *Note that only one instance of EVE may be running on a given database instance at any one time! If by some mistake more than one instance gets started, stop them both immediately and then run only one instance once the Validator Threads have cleared.*

Starting EVE

`ari_eve_process.start_eve` adds the `schedule_jobs` procedure (EVE itself) to `DBA_JOBS`. This procedure must be run as the ARI master user to ensure that it has the appropriate privileges when it is executed by the `DBA_JOBS` queue manager process. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Stopping EVE

`ari_eve_process.stop_eve` removes the `schedule_jobs` procedure (EVE itself) from the `DBA_JOBS` queue, and signals to the running process that it should terminate. The EVE process sends terminate signals to its Validator Threads and then terminates itself. This procedure must be run as the ARI master user to ensure that it has the appropriate privileges when it executes in the queue. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Periodic Purges

ARI uses a significant amount of data in two special areas, tracking events and tracking event history. Closed events that do not have a time-to-live (meaning almost all events – see the Online Help for reasons to create non-zero time-to-live events) are immediately removed from the system once they are closed. Events that do have a non-zero time-to-live must be periodically purged or they will remain in the system in a closed state indefinitely. Event History is retained after the event itself is closed for at least as long as is specified in the event type definition, and is only removed by running the history purge process.

Event Purge

Because there will not typically be very many events with a non-zero time-to-live, the need to periodically purge them is small, so the appropriate function `ari_event_instance_sql.delete_expired` need not be scheduled very often. On the other hand, because not many records will need purged it should not take very long at all if run often, so it could equally be scheduled to run daily or weekly or even less often depending on the number and kinds of events in your system.

Event History Purge

Event history should probably be purged daily or at least weekly. Times to execute will vary based on volume of events, but presumably in full production the reason for setting a history interval is to somehow regulate table sizes. For this to work correctly the history tables themselves will need the periodic purging of event history that is older than the number of retention days specified in the event type definitions. The function to schedule for history purging is `ari_event_hist_sql.purge_expired`.

ARI Alert Notification API

The purpose of this API is to enable displaying a button on the toolbar of an application other than ARI itself. The ARI Alert Viewer button serves two functions: as a gateway for launching the alert viewer and as a visual indicator of the overall contents of the alert viewer. Launching the alert viewer simply means opening that Form. The visual indicator is that the button changes appearance depending on whether or not new alerts exist.

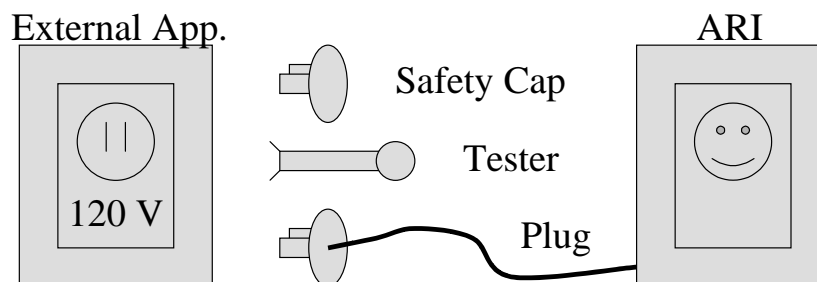
The API is implemented as a Forms library, a package and a table. The Forms library may not be appropriate if the application needing to interface with ARI is not a Forms application, but the substitution of that part of the API for a non-Forms application is an implementation issue not addressed by the product.

End User Cases

From the end-user standpoint this API has the following cases:

- (1) If the user is an ARI user with no new alerts the button should be displayed on the toolbar with the mailbox closed icon (ari_ari0.ico).
- (2) If the user is an ARI user with new alerts the button should be displayed on the toolbar with the mailbox open icon (ari_ari6.ico).
- (3) If the user is not an ARI user the button is not displayed.
- (4) In either scenario 1 or 2 the button should be pressed to attempt to launch the alert viewer. (This will display an error message if ARI is not truly installed.)
- (5) Switching from either scenario 1 or 2 to scenario 3 (removing a user as an ARI user) the button will remain until the Form containing the button is closed and re-launched (which should give scenario 3). Attempting to launch the alert viewer from this button after being removed as a user will give an error. This seems annoying but is acceptable because removing an ARI user is very difficult unless the user has no events, and almost never happens in production unless a user has no events. Moreover, by simply exiting and re-launching the form the error is self-correcting.
- (6) Switching from 3 to 1 or 2 the button will not appear until the Form containing the button is closed and re-launched.
- (7) Switching between 1 and 2 the icon should change accordingly when focus is changed from the window and back again. Minimizing and then reselecting a window is one way to do this.

Architecture



ARI provides an API that gets installed in the external Forms application. This API is like a socket. The socket is only able to accept specific values and the ARI plug is constrained to only deliver those specific values.

ARI also provides a tester module that can be configured to simulate all three end-user cases so the external application can test application behavior for all potential plug outputs/ARI interactions. The tester configured to drive the third end-user case (so that no user is an ARI user) is essentially the safety cap mode that will be deployed if the external application, in spite of installing the socket, is never deployed with ARI. This is the socket that is already installed in Retek's Merchandise Transaction System as mentioned in the Process Details section of this document.

When ARI is implemented the safety cap is replaced with a plug into ARI. This plug must have been tested in ARI to conform to the API specification that returns only three modes of operation corresponding to the first three end-user cases (the fourth case is an extension of one of the first two).

Implementation

Putting this all together, the socket part of the API is the Forms library. `ariiflib.pll` was installed in MTS (That version is full compatible with the other components of the API and does not need to be replaced with the latest version `ariif90.pll` that is shipped with ARI for use with any other (non-MTS) applications.) The tester part is the stub version of the `ari_interface_sql` package and the `ari_interface_test` table. To plug this into ARI, the non-ARI application simply drops its `ari_interface_sql` package and uses a synonym to the ARI master schema version of this package instead. *Note that localized modification of this package body for more sophisticated button icon display criteria or entirely different return values (requires more API modification) is allowed in spite of statements elsewhere in this documentation that no modifications of this product are supported.*

To install the socket the external application must add a button to the toolbar (or wherever they want a button) and must invoke the `GET_IS_ARI_USER` function to determine, based on whether the user is an ARI user, whether to display the button or enable its operation. This function should be installed so that the decision whether to display/enable the button is made with some frequency, depending on whether users are actively being added to and removed from the list of ARI users. Adding and removing users is non-trivial so the required frequency for such an action is minimal.

On some event-driven or periodic basis, including immediately after the button is displayed/enabled, the external application must invoke the `REFRESH_ALERT_ICON` function to update the button's displayed icon. The recommended implementation of this function is when the window focus changes (in the Forms When-Window-Activate trigger).

On pressing the alert viewer button the `LAUNCH_ALERT_VIEWER` procedure must be called. If ARI is installed on the same version of Forms as the application with which it will be working, this is a simple `OPEN_FORM` call. Otherwise, this is a remote call through the plug to tell ARI to launch, so the actual launching is external to the non-ARI application.

Testing Scenarios

The following test scenarios are presented to assist with testing how the non-ARI application will behave with the ARI Notification API installed before ARI is actually “plugged-in.”

The ARI_INTERFACE_TEST table controls the function of the tester, and by changing values in this table we can test the end-user scenarios in the external application. The table must contain only one row of data at a time and it has two columns. To exercise a certain a given scenario, the table must contain the values as shown. If the value changes mid-scenario that is indicated by an arrow →. Expected results for each scenario are understood by cross-referencing and understanding the seven end-user cases previously presented. Note also that the is_user_ind acts to globally make all users be, or not be, ARI users, which is sufficient for testing the external application behavior of most external applications we have imagined.

Scenario \ Test Table Column	IS_USER_IND	NEW_ALERT_IND
1. ARI user without new alerts	Y	N
2. ARI user with new alerts	Y	Y
3. Not an ARI user (safety)	N	N
4a. Launch alert viewer case 1	Y	N
4b. Launch alert viewer case 2	Y	Y
5a. Switch from 1 to 3 and launch alert viewer (gives error)	Y → N	N
5b. Switch from 2 to 3 and launch alert viewer (gives error)	Y → N	Y → N
5c. Switch from 1 to 3 and close and reopen Form	Y → N	N
5d. Switch from 2 to 3 and close and reopen Form	Y → N	Y → N
6a. Switch from 3 to 1 and do nothing (no button appears)	N → Y	N
6b. Switch from 3 to 2 and do nothing (no button appears)	N → Y	N → Y
6c. Switch from 3 to 1 and close and reopen Form	N → Y	N
6d. Switch from 3 to 2 and close and reopen Form	N → Y	N → Y
7a. Switch from 1 to 2 and change window focus or minimize	Y	N → Y
7b. Switch from 2 to 1 and change window focus or minimize	Y	Y → N

Appendix C – ARI Options

The following ARI configuration options are on the ARI options table. These configuration options are critical settings that affect the performance and processing of ARI. However, once they are set, they can largely be ignored. These options are set by a database administrator using Oracle SQL*Plus doing simple option value updates based on option name.

EVE_NUM_THREADS

This is the number of realm queue processes the validation engine should run at any one time. Generally this is the number of CPUs in the machine + 1, but not more than 32.

EVE_QUEUE_REFRESH_INTERVAL

This is the time slice in minutes given to each realm queue process. 10 minutes is recommended. It may need to be shorter to get reasonable real-time notification of exceptions if there are some very large queues and some very small ones. On the other hand, if the number of large queues is small enough, the entire time slice may only be used by one or two queues. If so, several processes will always be available for other processes, so a longer limit may improve the efficiency of processing those larger queues.

INTERNAL_SCHEMA

This is the name of the ARI user that is supposed to be created with the main ARI schema. This user has several explicit grants as described in the installation guide. It is in this schema that the ARI generated code and tables reside.

MASTER_SCHEMA

This is the main ARI schema. All of the static ARI code and tables are owned by this user, as described in the ARI Installation Guide.

MAX_EVENT_RECURSION

This value determines how many times an event will reevaluate itself as the result of auto-actions occurring before it raises an error. The value depends on the complexity of the states an auto-action structure of the defined event types. 5 is a reasonable value and no adjustment should be necessary except in the unlikely case of an event that needs to reevaluate legitimately (not though a mistake in the event definition) more than 5 times is created.

REEVAL_STATUS_LOCKOUT

This is the number of minutes between attempts to reevaluate events that could not be reevaluated either because they are locked or because their data values from remote data sources could not be fetched (e.g. because of a down database link).

PRIMARY_LANGUAGE_NUMBER

This is the number that identifies the language that will be considered the primary language for a multi-language ARI implementation. The primary language is also the only language in a single language implementation. The default value is based on the assumption that customers will use value 1 to represent their primary language.

ANALYST_ADMIN_GROUP_ID

This is here for reference by the ARI programs and should not be changed.

CLOSE_EVENT_REALM_ID

This is here for reference by the ARI programs and should not be changed unless the metadata for the close event action is altered by some mistake.

ERROR_ADMIN_GROUP_ID

This is here for reference by the ARI programs and should not be changed.

EVENT_INSTANCE_PARM_ID

This is here for reference by the ARI programs and should not be changed.

EXCEPTION_CREATE_DATE_PARM_ID

This is here for reference by the ARI programs and should not be changed.

EXCEPTION_CREATE_DATE_USER_ID

This is here for reference by the ARI programs and should not be changed.

LOG_LEVEL

Determines the log level of messages to be written to the ARI_ACTIVITY_LOG table. Valid levels are 1, 2 and 3. 3 is the most verbose logging and 1 only logs at a very high level. The default setting is 2, though this also may be unnecessarily verbose in a production environment.

FORWARD_GENERATION_HOURS

The value is a number of hours and determines how far ahead event and exception support code is generated before the definitions specify they are to become active. A detailed discussion exists in an earlier section on code generation. The default option is 30 hours allows some leeway if the code generator is scheduled to run daily.

An alternative to routinely scheduled generation, supplemented by the occasional on-demand generation, might be to generate *only* at the business analyst's request, but the business analyst must then be careful that everything new start within the generation window or else it may not get generated. Starting something far in the future either requires a large generation window or a periodic generation so that, as the start time approaches, the analyst does not risk forgetting that the code generator (if it is not run periodically) needs to be run.

The downside of a large generation window is that by generating far into the future, you cannot continue to modify a definition during the several days preceding its activation as you could with a 30-hour window and a daily code generation. To render such future-generated, no longer modifiable rules inactive, set the end date equal to the start date and regenerate code before the start date passes. Note however that, in production, constant re-tweaking before deployment should not be necessary since all such issues should have been worked out in development, so the definitions going into the production instance should have a solid foundation and a large generation window may be a perfectly fine approach. The issue is more significant in a development environment.

RDW_LINK

This option should contain the name of a database link to the database containing the schema named in the RDW_OWNER option (see below). This database link must be accessible to the ARI master schema and users of the Metadata form. The link should connect to a user with the "select any table" system privilege. If the RDW_OWNER schema is located in the same database as the ARI Master schema, then this option should be null.

RDW_OWNER

This option should contain the name of the RDW user who will own reports generated for ARI.

RDW_PREFIX

This option should contain the prefix that will be used to identify the tables in the Datamart schema that will contain cached report data. This option should be null if no common prefix is being used for Datamart tables.